To prove by induction that `PathFinder` computes the shortest paths and shortest-path distances from the source vertex `s` to each vertex `v` in the graph, we need to show that the breadth-first search (BFS) implemented in `PathFinder` correctly computes the shortest path for every vertex reachable from `s`.

Base Case (Step 0 of the BFS):

For the base case, consider vertices at distance 0 from the source.

- The only vertex at distance 0 is the source vertex itself.
- The source vertex `s` is enqueued into the `queue`, and its distance is set to `0`.
- In the constructor, we initialize `dist.put(s, 0)` for the source vertex s.
- This correctly represents that the distance to the source is 0, and the path to itself is trivial.
- **BFS Initialization:**
  - `queue.enqueue(s);`
  - `dist.put(s, 0);`

At this point, the only vertex in the queue is `s`, and the shortest distance from `s` to itself is correctly computed as `0`.

Thus, the base case holds: the shortest path from `s` to itself is correctly initialized to 0, and no other vertices are in the queue.

Inductive Hypothesis:

Assume that after `k ≥ 0` steps of BFS, the algorithm has correctly computed the shortest paths and shortest-path distances from the source `s` to all vertices that are at distance `k` from `s`.

Inductive Step (Step k+1 of the BFS):

We need to show that after `k + 1` steps of the BFS, the algorithm correctly computes the shortest paths and distances for all vertices at distance `k + 1` from `s`.

1. **At the beginning of the (k+1)-th step:**
   - All vertices at distance `k` from `s` have been dequeued from the `queue`, and for each of these vertices, all its adjacent vertices that have not yet been visited (i.e., not in `dist`) have been enqueued. For these enqueued vertices, their distance from `s` has been set to `k + 1`.

- This is guaranteed by the following code inside the BFS loop:

```
for (String w : G.adj(v)) {
    if (!dist.contains(w)) {
        queue.enqueue(w);
        dist.put(w, 1 + dist.get(v));  // Distance to w is set
to 1 + distance to v
        prev.put(w, v);                // Track the previous
vertex
    }
}
```

Consider a vertex `v` that is exactly `k+1` edges away from the source `s`. Let `u` be the vertex that precedes `v` on the shortest path from `s` to `v`. By our inductive hypothesis, `u` is `k` edges away from `s`, and its shortest path and distance have been correctly computed.

2. **Distance Computation**:

   - When the BFS reaches `u`, it explores all of `u`'s neighbors, including `v`.
   - If `v` hasn't been visited yet `(!dist.contains(w))`, it sets: `dist.put(w, 1 + dist.get(v));`
   - This computes the distance to `v` as 1 plus the distance to `u`.
   - Since `u` is on the shortest path to `v`, and the graph is unweighted, this distance is correct.

3. **Path Computation**:

   - When `v` is discovered through `u`, the algorithm sets: `prev.put(w, v);`
   - This correctly records `u` as the predecessor of `v` on the shortest path.
   - The `pathTo` method then reconstructs this path by following the `prev` links.

4. **Optimality**:

   - BFS explores vertices in order of their distance from the source.
   - When `v` is first discovered, it must be through a shortest path, as any longer path would have been explored later.
   - Once `v` is added to the queue, its distance is set and never changed, ensuring the first (shortest) path is preserved.

Therefore, for vertices at distance `k+1`, the algorithm correctly computes both the shortest path and the shortest-path distance.

2. **BFS Property:**
   - BFS explores all vertices at the same distance before moving on to vertices at the next greater distance. This ensures that when a vertex `w` is enqueued, it is the first time it is encountered, and its distance from `s` is the shortest path length. Once a vertex is dequeued, it is fully processed, meaning the shortest path to that vertex has already been found and no shorter path will be discovered.

- The algorithm does not re-enqueue any vertex once its distance has been set, which ensures that the first time a vertex is processed, the shortest path from `s` to that vertex is found.

Thus, after `k + 1` steps, all vertices at distance `k + 1` from `s` will be correctly processed, and their shortest distances will be set.

Conclusion (Inductive Proof):

By induction, after the BFS completes, the algorithm has computed the shortest path and shortest-path distance from the source vertex `s` to every vertex reachable from `s`. This holds because:

- The BFS processes vertices level by level (i.e., by increasing distance from the source).
- The first time a vertex is encountered in the BFS, the path leading to it is the shortest possible path.

Thus, the `PathFinder` class correctly computes the shortest paths and distances from the source `s` to each vertex in the graph.